Modeling Constrained Preemption Dynamics Of Transient Cloud Servers

Anonymous Author(s)

ABSTRACT

Scientific computing applications are being increasingly deployed on cloud computing platforms. Transient servers can be used to lower the costs of running applications on the cloud. However, the frequent preemptions and resource heterogeneity of these transient servers introduces many challenges in their effective and efficient use. In this paper, we develop techniques for modeling and mitigating preemptions of transient servers, and present SciSpot, a software framework that enables low-cost scientific computing on the cloud. SciSpot deploys applications on Google Cloud preemptible virtual machines, and introduces the first empirical and analytical model for their preemptions.

SciSpot's design is guided by our observation that many scientific computing applications (such as simulations) are deployed as "bag" of jobs, which represent multiple instantiations of the same computation with different physical and computational parameters. For a bag of jobs, SciSpot finds the optimal transient server on-the-fly, by taking into account the price, performance, and preemption rates of different servers. SciSpot reduces costs by 5× compared to conventional cloud deployments, and reduces makespans by up to 10× compared to conventional high performance computing clusters.

1 INTRODUCTION

Transient cloud computing is an emerging and popular resource allocation model used by all major cloud providers, and allows unused capacity to be offered at low costs as preemptible virtual machines. Transient VMs can be unilaterally revoked and preempted by the cloud provider, and applications running inside them face fail-stop failures. Due to their volatile nature, transient VMs are offered at steeply discounted rates. Amazon EC2 spot instances [?], Google Cloud Preemptible VMs [?], and Azure Batch VMs [?], are all examples of transient VMs, and are offered at discounts ranging from 50 to 90% compared to conventional, non-preemptible "on-demand" VMs.

To expand the usability and appeal of transient VMs, many systems and techniques have been proposed that seek to ameliorate the effects of preemptions and reduce the computing costs of applications. Fault-tolerance mechanisms, resource management policies, and cost optimization techniques have been proposed for a wide range of applications—ranging from interactive web services, distributed data processing, parallel computing, etc. These techniques have been shown to minimize the performance-degradation and downtimes due to preemptions, and reduce computing costs by up to 90%.

However, the success of these techniques depends on probabilistic estimates of when and how frequently preemptions occur. For instance, many fault-tolerance and resource optimization policies are parametrized by the mean time to failure (MTTF) of the transient VMs. For example, periodic checkpointing is a common technique for reducing the work lost due to preemptions, and the "optimal" checkpointing frequency that minimizes the total expected running time of a job depends on the MTTF of the VMs.

, ,
.

Past work on transient computing has focused on Amazon EC2's spot instances, whose preemption characteristics are determined by dynamic prices (which are in turn set using a continuous second-price auction). Transiency-mitigation techniques such as VM migration [?], checkpointing [? ?], diversification [?], *all* use price-signals to model the availability and preemption rates of spot instances. However, these pricing-based models are not generalizable to other transient VMs having a flat price (such as Google's or Azure's offerings). In these cases, the lack of information about preemption characteristics precludes most failure modeling and transient computing optimizations.

To address this gap, we seek to understand the preemption characteristics of Google's Preemptible VMs, whose distinguishing characteristic is that they have a *maximum lifetime of 24 hours*. We conduct a large empirical study of over 1,500 preemptions of Google Preemptible VMs, and develop an analytical probability model of preemptions. We find that the temporal constraint is a radical departure from pricing-based preemptions, and presents fundamental challenges in preemption modeling and effective use of Preemptible VMs.

Due to the temporal preemption constraint, classical models that form the basis of preemption modeling and policies, such as memoryless exponential failure rates, are not applicable. We find that preemption rates are *not* uniform, but bathtub shaped with multiple distinct temporal phases, and are incapable of being modeled by existing bathtub distributions such as Weibull. We capture these characteristics by developing a new probability model. Our model uses reliability theory principles to capture the 24-hour lifetime of VMs, and generalizes to VMs of different resource capacities, geographical regions, and across different temporal domains. To the best of our knowledge, this is the *first* work on constrained preemption modeling. Our investigation also points to a new, surprising connection to statistical mechanics, which can lead to new insights for modeling temporally constrained events.

We show the applicability and effectiveness of our model by developing optimized policies for job scheduling, checkpointing, and server selection. These policies are fundamentally dependent on empirical and analytical insights from our model such as different time-dependent failure rates of different types of VMs. These optimized policies are a building block for transient computing systems and reducing the performance degradation and costs of preemptible VMs. We implement and evaluate these policies as part of a new software framework for running scientific computing applications, called Our service.

Our service abstracts typical scientific computing workloads and workflows into a new unit of execution, which we call as a "bag of jobs". These bags of jobs, ubiquitous in scientific computing, represent multiple instantiations of the same application launched with possibly different physical and computational parameters. The bag of jobs abstraction permits efficient implementation of our optimized policies, and allows Our service to lower the costs and barriers of transient VMs for scientific computing applications.

Towards our goal of developing a better understanding of constrained preemptions, we make the following contributions:

(1) We develop a probability model of constrained preemptions based on a large-scale, first-of-its-kind empirical study of lifetimes of Google Preemptible VMs for understanding and characterizing their preemption dynamics. Our model captures the key effects resulting from the 24 hour lifetime constraint associated with these VMs, and we analyze it through the lens of reliability theory and statistical mechanics.
(2) We develop optimized policies for job scheduling, periodic checkpointing, and VM selection, that minimize the total time and cost of running applications. These policies are based on our preemption models, and reduce job running times by up to 40% compared to existing preemption models used for transient VMs.
(3) We implement all our policies as part of a new framework, Our service, and evaluate the cost and performance of different representative scientific computing applications on the Google Cloud Platform. Compared to conventional cloud deployments, Our service can reduce costs of running bags of jobs by more than 5×, and when compared to dedicated HPC clusters, it can reduce the total turnaround time by up to an order of magnitude.

2 BACKGROUND

We now give an overview of transient cloud computing, and motivate the need for the bag of jobs abstraction in scientific computing workflows.

2.1 Transient Cloud Computing

Infrastructure as a service (IaaS) clouds such as Amazon EC2, Google Public Cloud, Microsoft Azure, etc., typically provide computational resources in the form of virtual machines (VMs), on which users can deploy their applications. Conventionally, these VMs are leased on an “on-demand” basis: cloud customers can start up a VM when needed, and the cloud platform provisions and runs these VMs until they are shut-down by the customer. Cloud workloads, and hence the utilization of cloud platforms, shows large temporal variation. To satisfy user demand, cloud capacity is typically provisioned for the *peak* load, and thus the average utilization tends to be low, of the order of 25% [**? ?**].

To increase their overall utilization, large cloud operators have begun to offer their surplus resources as low-cost servers with *transient* availability, which can be preempted by the cloud operator at any time (after a small advance warning). These preemptible servers, such as Amazon Spot instances [**?**], Google Preemptible VMs [**?**], and Azure batch VMs [**?**], have become popular in recent years due to their discounted prices, which can be 7-10× lower than conventional non-preemptible servers. Due to their popularity among users, smaller cloud providers such as Packet [**?**] and Alibaba [**?**] have also started offering transient cloud servers.

However, effective use of transient servers is challenging for applications because of their uncertain availability [**? ?**]. Preemptions are akin to fail-stop failures, and result in loss of the application's memory and disk state, leading to downtimes for interactive applications such as web services, and poor throughput for long-running batch-computing applications. Consequently, researchers have explored fault-tolerance techniques such as checkpointing [**? ? ?**] and resource management techniques [**?**] to ameliorate the effects of preemptions. The effect of preemptions is dependent on a combination of application resource and fault model, and mitigating preemptions for different applications remains an active research area [**?**].

2.2 Preemption Modeling of Transient VMs

Past work on transient computing has almost exclusively focused on the Amazon EC2 spot market, which has a radically different preemption model and dynamics compared to other transient cloud servers such as those offered by Google Cloud and Azure.

Launched in 2009, Amazon's EC2 spot instances are the first example of transient cloud servers. The preemptions of EC2 spot instances are based on their *price*, which is dynamically adjusted based on the supply and demand of cloud resources. Spot prices are based on a continuous second-price auction, and if the spot price increases above a pre-specified maximum-price, then the server is preempted [**?**].

Thus, the time-series of these spot prices can be used for understanding preemption characteristics such as the frequency of preemptions and the “Mean Time To Failure” (MTTF) of the spot instances. Publicly available[1]historical spot prices have been used to characterize and model spot instance preemptions [**? ?**]. For example, past work has analyzed spot prices and shown that the MTTFs of spot instances of different hardware configurations and geographical zones range from a few hours to a few days [**? ? ?**]. (Prateek: sigcomm paper)

However the price-based approach for inferring preemption characteristics is of limited utility. Other transient VMs such as those offered by Google and Azure have *flat* pricing, and price-based techniques for preemption modeling and cost optimization are not applicable. Furthermore, price-based techniques may be of limited value after Amazon's recent change where preemptions were decoupled with pricing. To address this shortcomings, we develop *empirical* preemptions models for transient VMs. In particular, we focus on Google Preemptible VMs, whose preemption properties have been unknown so far. Additionally, their unique fixed lifetime leads to new challenges for conventional reliability theory models, and requires new modeling techniques, which we develop later in Section **??**.

3 CONSTRAINED PREEMPTIONS OF GOOGLE PREEMPTIBLE VMS

To measure and improve the performance of applications running on transient cloud servers, it is critical to understand the nature and dynamics of their preemptions. In this section, we present empirical analysis of preemptions of Google Preemptible VMs, develop a new empirically-informed probability model that predicts preemptions, and discuss the unique aspects of the observed preemption rates with a simple analytical calculation.

[1]Amazon posts Spot prices of 3 months, and researchers have been collecting these prices since 2010 [**?**].

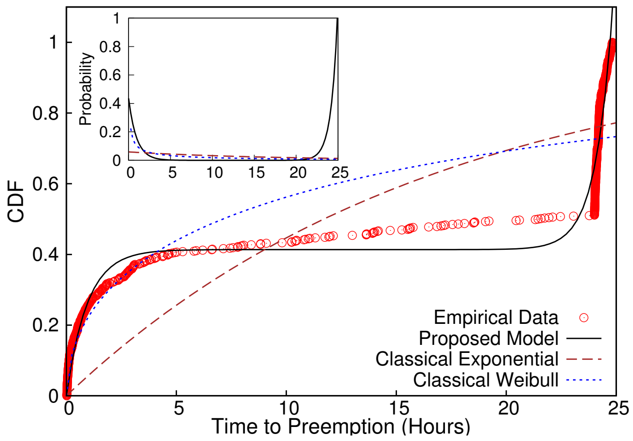


Figure 1: CDF of lifetimes of Google Preemptible VMs. Our proposed distribution for modeling the constrained preemption dynamics provides better fits to the empirical data compared to the conventional exponential and the Weibull distributions. Inset shows the probability of failure as a function of time for the three distributions.

3.1 Empirical Study Of Preemptions

To understand the nature of temporally constrained preemptions, we conducted the first empirical study of Google's Preemptible VMs, that have a fixed price and a maximum 24 hour lifetime. Our empirical study is necesitated by the fact that the cloud operator (Google) does not disclose any other information about the preemption rates, and thus relatively little is known about the preemptions of these VMs, and as a result their performance.

We launched more than 1,500 Google Preemptible VMs of different types over a two month period (Feb–April 2019), and measured their time to preemption (i.e., their useful lifetime).[2] A sample of over 100 such preemption events are shown in Figure 1, which shows cumulative distribution function (CDF) of the VM lifetimes of the `n1-highcpu-16` VM in the `us-east1-b` zone. Note that the cloud operator (Google) caps the *maximum* lifetime of the VM to 24 hours, and all the VMs are preempted before that limit.

Observation 1: *The lifetimes of VMs are not uniformly distributed, but have three distinct phases.*

In the first (initial) phase, characterized by VM lifetime $t \in [0, 3]$ hours, we observe that many VMs are quickly preempted after they are launched, and thus have a steep rate of failure. The rate of failure (preemption rate) is the derivative of the CDF. In the second phase, VMs that survive past 3 hours enjoy a relatively low preemption rate over a relatively broad range of lifetime (characterized by the slowly rising CDF in Figure 1). The third and final phase exhibits a steep increase in the number of preemptions as the preemption deadline of 24 hours approaches. The overall rate of preemptions is "bathtub" shaped as shown by the solid black line in the inset of Figure 1 (discussed in detail below).

[2]We will release the complete preemption dataset for further analysis.

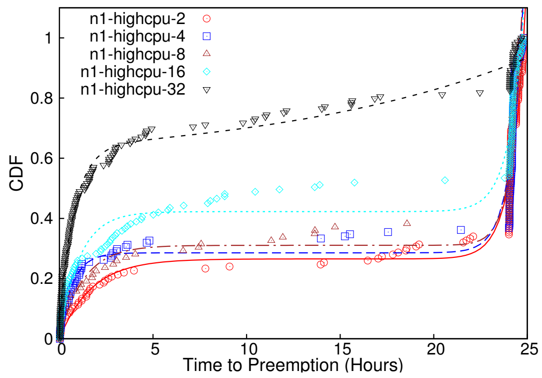


Figure 2: Preemption characteristics of different VM types. Larger VMs are more likely to be preempted.

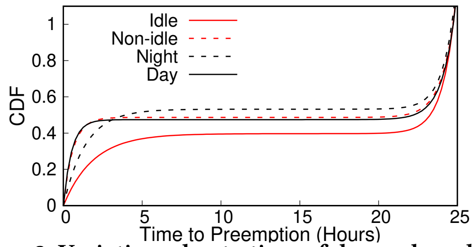


Figure 3: Variations due to time of day and workload.

Observation 2: *The preemption behavior, imposed by the constraint of the 24 hour lifetime, is substantially different from conventional failure characteristics of hardware components and EC2 spot instances.*

In "classical" reliability analysis, the rate of failure usually follows an exponential distribution $f(t) = \lambda e^{-\lambda t}$, where $\lambda = 1/\text{MTTF}$. Figure 1 shows the CDF ($= 1 - e^{-\lambda t}$) of the exponential distribution when fitted to the observed preemption data, by finding the distribution parameter λ that minimizes the least squares error. The classic exponential distribution is unable to model the observed preemption behavior because it assumes that the rate of preemptions is independent of the lifetime of the VMs, i.e., the preemptions are *memoryless*. This assumption breaks down when there is a fixed upper bound on the lifetime, as is the case for Google Preemptible VMs.

Observation 3: *The three preemption phases and associated bathtub shaped preemption probability are general, universal characteristics of Preemptible VMs.*

In general, the preemption dynamics of a VM is determined by the supply and demand of VMs of that *particular* type. Thus, our empirical study looked at preemptions of VMs of different sizes, in different geographical zones, at different times of the day, and running different workloads (Figure **YYY**). In all cases, we find that there are three distinct phases associated with the preemption dynamics giving rise to the bathtub shaped preemption probability. We argue that this is not a coincidence, but may be a result of practical and fundamental outcomes of cluster management policies.

While the actual specific preemption policy is up to the cloud operator, we will show that the bathtub behavior has benefits for applications. For applications that do not incorporate explicit fault-tolerance (such as checkpointing), early preemptions result in less wasted work than if the preemptions were uniformly distributed over the 24 hour interval. Furthermore, the low rate of preemptions in the middle periods allows jobs that are smaller than 24 hours to finish execution with only a low probability of failure, once they survive the initial preemption phase. We compare job performance with bathtub shaped preemptions vs. uniform preemption rates in Section Evaluation.

~~In addition to being beneficial to applications, we also conjecture that the bathtub behavior is a *fundamental* characteristic of any system where events are randomly distributed in a finite interval. Intriguingly, we can analyze such temporally constrained preemptions through a statistical mechanics framework, and we elaborate on this connection later in Section ??.~~

Observation 4: *Larger VMs have a higher rate of preemptions.* Figure 2 shows the preemption data from five different types of VMs in the Google Cloud `n1-highcpu-{2,4,8,16,32}`, where the number indicates the number of CPUs. All VMs are running in the `us-central1-c` zone. We see that the larger VMs (16 and 32 CPUs) have a higher probability of preemptions compared to the smaller VMs. While this could be simply due to higher demand for larger VMs, it can also be explained from a cluster management perspective. Larger VMs require more computational resources (such as CPU and memory), and when the supply of resources is low, the cloud operator can quickly reclaim a large amount of resources by preempting larger VMs. This observed behavior aligns with the guidelines for using preemptible VMs that suggests the use of smaller VMs when possible [?].

Observation 5: *Preemptions exhibit diurnal variations, and are also affected by the workload inside the VM.* From Figure 3, we can see that VMs have a slightly longer lifetime during the night (8 PM to 8 AM) than during the day. This is expected because fundamentally, the preemption rates are higher during periods of higer demand.

We also notice that completely idle VMs have longer lifetimes than VMs running some workload. Presumably, this could be a result of the lower resource utilization of idle VMs being more amenable to resource overcommitment, and result in lower preemptions.

3.2 Failure Probability Model

We now develop an analytical probability model for finding a preemption at time t (preemption dynamics) that is faithful to the empirically observed data and provides a basis for developing running-time and cost-minimizing optimizations. Modeling preemptions constrained by a finite deadline raises many challenges for existing preemption models that have been used for other transient servers such as EC2 spot instances. We first discuss why existing approaches to preemption modeling are not adequate, and then present our closed-form probability model and associated reliability theory connections.

3.2.1 Inadequacy of existing failure distributions. Spot instance preemptions have been modeled using exponential distribution [? ? ?], which is the default in most reliability theory applications. However, the strict 24 hour constraint and the distinct preemption phases are not compatible with the memoryless properties of the exponential distribution. To describe failures (preemptions) that are not memoryless (i.e., increasing or decreasing failure rate over time), the classic Weibull distribution with CDF $F(t) = 1 - e^{-(\lambda t)^k}$ is often employed. However, the Weibull distribution is also unable to fit the empirical data (Figure 1) and especially unable to model the sharp increase in preemptions near the 24 hour deadline.

For constrained preemptions, the increase in failure rate as modeled by the Weibull distribution is not high enough. Other distributions, such as Gompertz-Makeham, have also been used for modeling bathtub behavior, especially for actuarial use-cases. The key idea is to incorporate an exponential aging process, which is used to model human mortality. The CDF of the Gompertz-Makeham distribution is given by $F(t) = 1 - \exp\left(-\lambda t - \frac{\alpha}{\beta}(e^{\beta t} - 1)\right)$ and is fitted to the data in Figure XXX, and is also unable to provide a good model for the observed preemption data.

The non-trivial bathtub-shaped failure rate of Google preemptible VMs (Figure 1) requires models that capture the sudden onset of the rise in preemptions near the deadline. From an application and transiency policy perspective, the preemption model must provide insights about the phase transitions, so that the application can adapt to the sharp differences in preemption rates. For example, the preemption model should be able to warn applications about impending deadline, which existing failure distributions cannot account for. Thus, not only is it important to minimize the total distribution fitting error, it is also important to capture the changes in phase. However, as we can see from the **QQ plots**, existing distributions are unable to capture the effects of the deadline and the phases of the preemptions, and a new modeling approach is needed, which we develop next.

3.2.2 Multiple failure rate model. Our failure probability model seeks to address the drawbacks of existing reliability theory models for modeling constrained preemptions. The presence of three distinct phases exhibiting non-differentiable transition points (sudden changes in CDF near the deadline, for example) suggests that for accurate results, models that treat the probability as a step function (CDF as a piecewise-continuous function) should be employed. However, this limits the range of model applicability and general interpretability of the underlying preemption behavior. Our goal is to provide a broadly applicable, (Vikram: continuosly differentiable), and informative model built on reasonable assumptions.

We begin by making a key assumption: the preemption behavior arises from the presence of *two* distinct failure processes. The first process dominates over the initial temporal phase and yields the classic exponential distribution that captures the high rate of early preemptions. The second process dominates over the final phase near the 24 hour maximum VM lifetime and is assumed to be characterized by an exponential term that captures the sharp rise in preemptions that results from this constrained lifetime.

Based on these observations, we propose the following general form for the CDF:

$$\boxed{\mathscr{F}(t) = A\left(1 - e^{-\frac{t}{\tau_1}} + e^{\frac{t-b}{\tau_2}}\right)} \quad (1)$$

where t is the time to preemption, $1/\tau_1$ is the rate of preemptions in the initial phase, $1/\tau_2$ is the rate of preemptions in the final phase, b denotes the time that characterizes "activation" of the final phase where preemptions occur at a very high rate, and A is a scaling constant.

The temporal interval or deadline of 24 hours, noted with the parameter L, enters implicitly by demanding that $0 < t < L$. Combination of the 4 fit parameters (τ_1, τ_2, b, and A) are chosen to ensure that boundary conditions are reasonably met: $\mathscr{F}(0) \approx 0$ and $\mathscr{F}(L) \approx 1$. When not used as a fit parameter, A is chosen as $A = 1/(1 - e^{-\frac{L}{\tau_1}} + e^{\frac{L-b}{\tau_2}})$ to ensure $F(L) = 1$, yielding results similar to the 4 parameter fit. In practice, typical fit values yield $b \approx 24$ hours, τ_1 =??, $\tau_2 \approx 1$ hour, and $A \approx 1/(1 - e^{-\frac{L}{\tau_1}} + e^{\frac{L-b}{\tau_2}}$. (Vikram: I suggest giving ranges here for all)

(Vikram: check, and provide typical fit values for all these parameters as well as their range extremes. ideally, you want to fit with A set to the above expression to avoid any $\mathscr{F}(t)$ or $p(t) > 1$ scenarios. turns out that having A is important in reliability analysis – check next column)

For most of its life, a VM sees failures according to the classic exponential distribution with a rate of failure equal to $1/\tau_1$ – this behavior is captured by the $1 - e^{-t/\tau_1}$ term in Equation 1. As VMs get closer to their maximum lifetime imposed by the cloud operator, they are reclaimed (i.e., preempted) at a high rate $1/\tau_2$, which is captured by the second exponential term, $e^{(t-b)/\tau_2}$ of Equation 1. Shifting the argument (t) of this term by b ensures that the exponential reclamation is only applicable near the end of the VM's maximum lifetime and does not dominate over the entire temporal range.

The analytical model and the associated distribution function $\mathscr{F}$ introduced above provides a much better fit to the empirical data (Figure 1) and captures the different phases of the preemption dynamics through parameters τ_1, τ_2, b, and A. These parameters can be obtained for a given empirical CDF using least squares function fitting methods. The failure or preemption rate can be derived from this CDF as:

$$f(t) = \frac{d\mathscr{F}(t)}{dt} = A\left(\frac{1}{\tau_1}e^{-t/\tau_1} + \frac{1}{\tau_2}e^{\frac{t-b}{\tau_2}}\right). \quad (2)$$

$p(t)$ vs. t yields a bathtub type failure rate function for the associated fit parameters (inset of Figure 1).

In the absence of any prior work on constrained preemption dynamics, our aim is to provide an interpretable model with a minimal number of parameters, that provides a sufficiently accurate characterization of observed preemptions data. Further generalization of this model to include more failure processes would introduce more parameters and reduce the generalization power. Exploring other approaches of modeling bathtub-type failure rates (e.g., exponential Weibull distributions) [? ?] is part of our future work.

3.2.3 Reliability Analysis. We now analyze and place our model in a reliability theory framework.

Expected Lifetime: Our analytical model also helps crystallize the differences in VM preemption dynamics, by allowing us to easily calculate their expected lifetime. More formally, we define the expected lifetime of a VM of type i, as:

$$E[L_i] = \int_0^{24} t f_i(t)\, dt = -A(t + \tau_1)e^{-t/\tau_1} + A(t - \tau_2)e^{\frac{t-b}{\tau_2}}\Big|_0^{24} \quad (3)$$

where $f_i(t)$ is the rate of preemptions of VMs of type i (Equation 2).

This expected lifetime can be used in lieu of MTTF, for policies and applications that require a "coarse-grained" comparison of the preemption rates of servers of different types, which is a key requirement in cost-minimizing server selection, which we develop later in Section.

Hazard Rate: The hazard rate $\lambda(t)$ governs the dynamics of the failure (or survival) processes. It is generally defined as $\lambda(t) = \frac{g(t)}{S(t)}$, often expressed via the following differential equation (rate law):

$$\frac{dS(t)}{dt} = -\lambda(t)S(t) \quad (4)$$

where $S(t) = 1 - F(t)$ is the survival function associated with a CDF $F(t)$, and $g(t) = dF(t)/dt$ is the failure probability function (rate) at time t. The survival function indicates the amount of VMs that have survived at time t. The hazard rate can also be directly expressed in terms of the CDF as follows: $1 - F(t) = \exp \int_0^t -\lambda(x)\, dx$. The exponential distribution has a constant hazard rate λ. The Gompertz-Makeham distribution has an increasing failure rate to account for the increase in mortality, and its hazard rate is accordingly non-uniform and given by $\lambda(t) = \lambda + \alpha e^{\beta t}$.

Since we model multiple failure rates and deadline-driven preemptions, our hazard rate is expected to increase with time. Defining the survival function for our model: $S = 1 - \mathscr{F}$, and using Eq. 4 yields the hazard rate associated with our model:

$$\lambda = \frac{r_1 e^{-r_1 t} + r_2 e^{r_2(t-b)}}{1/A - 1 + e^{-r_1 t} - e^{r_2(t-b)}} \quad (5)$$

where we have introduced $r_1 = 1/\tau_1$, $r_2 = 1/\tau_2$ to denote the rates of preemptions associated with initial and final phases respectively. (Vikram: without the A term, hazard rate becomes negative for the older expression you had when $t > br2/(r1 + r2)$. that is for t roughly greater than b/2, which is for more than 12 hours. hazard rate can never be negative.)

Employing the value for A resulting from ensuring (via fit or by force) that our CDF goes to 1 at $t = L$ (where L is 24 hours), we find

$$\lambda = \frac{r_1 e^{-r_1 t} + r_2 e^{r_2(t-b)}}{e^{r_2(L-b)} - e^{-r_1 L} + e^{-r_1 t} - e^{r_2(t-b)}} \quad (6)$$

Recall that the sharp increase in preemption rate only happens close to the deadline, which means that $b \lesssim L$. Thus, when $0 < t \ll b$, we get $\lambda(t) \approx r_1$, mimicking the hazard rate for the classic exponential distribution. As t approaches and exceeds b (i.e., $b \lesssim t < L$), the increase in the hazard rate due to the second failure process kicks in, accounting for the deadline-driven rise in preemptions. Note that our hazard rate satisfies $\lambda(t) \geq 0$ for $0 < t < L$.

3.3 Insights on the bathtub shape distribution

The immediate yet striking feature of the bathtub shaped probability distribution associated with events occuring in a constrained system is that the probability of finding an event (in our case the preemption) at the edge (deadline) is higher than the probability of finding an event at the center of the interval. We now show using exact analytical arguments that this aspect is a general feature of constrained systems assuming some assumptions are satisfied. We later discuss the justification of these assumptions for our specific problem of constrained preemptions.

Lemma: Consider N preemptions to occur randomly with a flat probability distribution within a finite temporal interval $0 < t < L$ such that the preemptions are mutually exclusive. We define $p(t)$ as the probability of finding a preemption at time t. Then, there exists an $\epsilon > 0$ such that $\frac{p(L-\epsilon)}{p(L/2)} > 1$.

Proof: We first make some preliminary remarks and introduce concepts necessary to complete the proof. Mutual exclusion of preemptions implies that there is a finite non-zero waiting time between preemptions. Let this waiting time be w. Thus N preemptions occupy a "temporal volume" of Nw. The amount of free volume or excluded volume available in the constrained system is $L - Nw$ given that L is the maximum possible volume (note that the term volume is used to mean one-dimensional volume). The idea of excluded volume is central in physics and materials engineering where it underpins the origin of entropic forces in material systems.

We begin by counting the total number of configurations available to place N preemptions in the constrained temporal "volume" L. The number of configurations one preemption can generate is proportional to the excluded volume available, i.e., $L - Nw$. For N preemptions, the number of configurations grow exponentially and are proportional to the volume of the N-dimensional hypercube: $Z = (L - Nw)^N$.

Given the condition of mutually exclusive preemptions and the interval definition, we note that if we assume that there is a preemption at $t = L - w$, then $p(t > L - w) = 0$. For this case, the available configurations are proportional to the temporal volume accessible in the $N-1$ dimensional hypercube to $N-1$ preemptions: $Z_1 = (L - w - (N-1)w)^{N-1} = (L - Nw)^{N-1}$.

The probability of finding 1 preemption near the deadline at $t = L - w$ is given as the ratio of the two computed volumes. That is, $p(L - w) = Z_1/Z = 1/(L - Nw)$.

Given the uniform distribution of the placement of N preemptions, the probability of finding a preemption near the middle of the interval at $t = L/2$ approaches $1/L$ as $L \to \infty$, that is $\lim_{t\to\infty} p(L/2) = 1/L$. We note that L is finite (given the notion of constraint). (Vikram: this can be made more precise from here on, but I am cheating now)

$p(L/2) = 1/L$ follows because we can always choose a large enough L to make $p(L/2) - 1/L$ as close to 0 as possible.

Choosing $\epsilon = w$ and observing that $1/(L - Nw) - 1/L > 0$ for every finite L and non-zero w, proves $p(L - \epsilon) > p(L/2)$ or $\frac{p(L-\epsilon)}{p(L/2)} > 1$.

3.4 Elsewhere: Role of Cloud Providers: Analogy to Constrained Physical Systems

The bathtub-shaped distribution of the preemption probability is essential to capture the empirical data with higher accuracy compared to other models. We want to probe the following question now: how much of this unique shape is enforced by the cloud provider, and how much is dictated by assuming that (akin to other cloud providers such as EC2), preemptions of Google Preemptible VMs are placed along the timeline of 24 hours with uniform probability? We make the question more precise and inject a relatively harmless assumption: if the cloud provider decides to "place" N preemptions within the temporal boundary of interval L with flat probability distribution ($\mathscr{P}(e_1, e_2, \ldots e_N) = 1$ whenever legal) such that the preemptions are mutually exclusive (that is there is a finite non-zero waiting time between preemptions), what is the probability of finding a preemption at time t? Naively, deducing from $\mathscr{P}$, one may expect $p(t)$ to be 1 for $t \in L$, or a constant (flat) distribution. However, intruigingly, this is not true. Thus, as we discuss in more detail below, the cloud provider may have the simplest of preemption policies implemented at their end, and yet give rise to the relatively complex and non-trivial probability to preemption.

To understand the above assertions with a simple example that sheds further insights into the origins of the bathtub-shaped distribution, we discuss analogous questions in the context of general physical systems of constrained (confined) particles. These systems are generally analyzed using the theoretical framework of statistical mechanics often complemented with associated particle dynamics simulations. We choose an exactly solvable and simple system of hard (non-overlapping) particles (or beads) confined to move in one dimension within a finite spatial interval to draw interesting connections.

A simple mapping from the system of temporally-constrained preemptions in the cloud to the system of spatially-constrained particles within one-dimensional confinement can be established by assuming that the cluster management policy requires mutually exclusive preemptions. This assumption helps cluster management by serializing VM preemptions and making cluster operations easier, and it reduces the challenges associated with reacting to simultaneous preemptions. The preemption events become hard particles that cannot overlap. The preemption probability distribution gets mapped to the density of N particles, each modeled as a one-dimensional "sphere" of width σ, confined in a 1-dimensional interval (line segment) of length L. L can be considered as corresponding to the deadline interval of 24 hours. σ is equivalent to the length of the critical section, which is related to the preemption warning (30 seconds for Google PVMs).

The particles are placed, one after another, at random positions. Configurations where any overlap is generated are not included. Thus particles will be placed with a flat probability distribution. The question we are trying to simulate with this set up is the familiar one noted above: what is the probability $p(x)$ of finding a preemption at time x?

This problem can be solved by first defining a partition function for the system. And doing some neat math.

Present the solution

This distribution is *not* uniform over the interval, but is higher towards the ends of the interval—analogous to the bath-tub shaped nature of preemptions!

Discuss more.

We thus draw a few critical observations. An immediate inclincation may be to attribute the observed non-trivial bathtub-shaped preemption probability function to a uniquely designed and structured operational policy of the cloud provider. However, the above analysis shows that even with the vanilla operational policy of placing preemptions with a uniform probability distribution, the finite excluded temporal volume of preemptions in conjunction with the relatively short deadline (which together imply a high average preemption rate) enforce a bathtub-shape-like distribution for finding a preemption at time t. Clearly, this does not provide all the subtle features observed in the preemption data as well as the failure model. But even there, it enables to gauge the extent to which the provider has changed the policy on top of the simplest possible implementation.

The analytical framework suggests that the bath-tub shape of the probability distribution is the key characteristic of constrained preemptions. Therefore, our approach will remain relevant in the face of future changes in preemption characteristics due to changes in cloud usage and operational policies.

4 APPLICATION POLICIES FOR CONSTRAINED PREEMPTIONS

For applications running on transient servers, the effects of preemptions can be ameliorated through different policies for fault tolerance and resource management. Prior work in transient computing has established the benefits of such policies for a broad range of applications. We now show how our empirical insights and analytical model can be used to develop policies for optimized execution of batch applications on Google Preemptible VMs.

4.1 Job Scheduling

Many cloud-based applications and services are *long-running*, and typically run a continuous sequence of tasks and jobs on cloud servers. In the case of deadline-constrained bathtub preemptions, applications face a choice: they can either run a new task on an already running server, or relinquish the server and run the task on a *new* server. This choice is important in the case of non-uniform failure rates, since the job's failure probability depends on the "age" of the server. Furthermore, since newly launched servers also have high preemption rates (and thus high job failure probability), the choice of running the job on an existing server vs. a new server is not obvious.

Our job scheduling policy uses the preemption model to determine the preemption probability of jobs of a given length T. Assume that the running server's age (time since launch) is s. Then, the the probability of failure on the existing server $P_{\text{Existing}} = max(1, F(T+s) - F(T))$. The alternative is to discard the server and launch a new server, in which case, the failure probability is $P_{\text{New}} = F(T)$. Depending on the server's age s and the job's running time T, we can compare P_{Existing} and P_{New}, and run the job on whichever case yields the lower failure probability.

4.2 Periodic Checkpointing

A common technique for reducing the total expected running time of jobs on transient servers is to use fault-tolerance techniques such as periodic checkpointing [**?**]. Checkpointing application state to stable storage (such as network file systems or centralized cloud storage) reduces the amount of *wasted work* due to preemptions. However, each checkpoint entails capturing, serializing, and writing application state to a disk, and increases the total running time of the application. Thus, the frequency of checkpointing can have a significant effect on the total expected running time.

Existing checkpointing systems for handling hardware failures in high performance computing, and for cloud transient servers such as EC2 spot instances, incorporate the classic Young-Daly periodic checkpointing interval that assumes that failures are exponentially distributed. That is, the application is checkpointed every $\tau = \sqrt{2 \cdot \delta \cdot \text{MTTF}}$ time units, where δ is the time overhead of writing a single checkpoint to disk.

However, checkpointing with a uniform period is sub-optimal in case of time dependent failure rates, and especially for bathtub failure rates. A sub-optimal checkpointing rate can lead to increased recomputation and wasted work, or result in excessive checkpointing overhead. Intuitively, the checkpointing rate should depend on the failure rate, and our analytical preemption model can be used for designing an optimized checkpointing schedule.

Let the uninterrupted running time of the job be T. Or in other words, T amount of work needs to be performed. Let the checkpoint cost be δ. We seek to minimize the total expected running time or the *makespan*, which is the sum of T, the expected periodic checkpointing cost, and the expected recomputation.

The makespan M can be recursively defined and computed. Let $M(W,t,i)$ denote the makespan when W length of job must be exectued, t is amount of job already done. We now need to determine when to take the *next* checkpoint, which we take after i steps. Let $E[M^*]$ denote the minimum expected makespan.

$$E[M^*(W,t)] = \min_{0<i\leq W} E[M(W,t,i)] \tag{7}$$

The makespan is affected by whether or not there is a failure *before* we take the checkpoint:

$$E[M(W,t,i)] = P_{\text{succ}}(t, i+\delta) \cdot E[M_{\text{succ}}] + P_{\text{fail}}(i+\delta, t) \cdot E[M_{\text{fail}}] \tag{8}$$

Here $P_{\text{succ}}(t, i+\delta)$ denotes the probability of the job successfully executing without failures until the checkpoint is taken, i.e., from t to $t+i+\delta$. $P_{\text{fail}}(t, i+\delta) = F(t+i+\delta) - F(i+\delta)$ is computed using the CDF, and $P_{\text{succ}} = 1 - P_{\text{fail}}$.

$E[M_{\text{succ}}]$ is the expected makespan when there are no job failures when the job is executing from step t to $t+i+\delta$, and is given by a recursive definition:

$$E[M_{\text{succ}}(W,t,i)] = t + i + \delta + E[M^*(W-i, t+i+\delta)] \tag{9}$$

Note that the makespan includes the amount of work already done $t+i$, the checkpointing overhead δ, and the expected minimum makespan of the rest of the job. Similarly, when the job fails before step i, then that portion is "lost work", and can be denoted by $E[L(t, i+\delta)]$ which denotes the expected lost work when there is a failure during the time interval t to $t+i+\delta$. A failure before the checkpoint means that we have made no progress, and W steps of the job still remain. The expected makespan in the failure case is then given by:

$$E[M_{\text{fail}}(W,t,i)] = E[L(t, i+\delta)] + E[M^*(W, t+i+\delta)] \tag{10}$$

In the case of memoryless distribution, $E[L(t, i+\delta)]$ is approximated as $\frac{i+\delta}{2}$. In our case, we use the failure CDF instead:

$$E[L(t, i+\delta)] = \int_{t}^{t+i+\delta} x f(x)\, dx, \tag{11}$$

where $f(x)$ is our probability density function represented by Equation 2.

Thus we can find the minimum makespan $E[M^*(W,t)]$ by using Equations 7–11. Given a job of length J, minimizing the total expected makespan involves computing $E[M^*(J,s)]$, where s is the current age of the server. Since the makespan is recursively defined, we can do this minimization using a dynamic programming approach, and extract the job-steps at which checkpointing results in a minimum expected makespan.

5 IMPLEMENTING A BATCH COMPUTING SERVICE FOR PREEMPTIBLE VMS

We have implemented a prototype batch computing service that implements various policies for constrained preemptions. We use this service to examine the effectiveness and practicality of our model and policies in real-world settings.

Our service is implemented as a light-weight, extensible framework that makes it convenient and cheap to run batch jobs in the cloud. We have implemented our prototype in Python in about 2,000 lines of code, and currently support running VMs on the Google Cloud Platform [?].

Our service is implemented as a centralized controller, which implements the VM selection and job scheduling policies described in Section **??**. The controller can run on any machine (including the user's local machine, or inside a cloud VM), and exposes an HTTP API to end-users. Users submit bags of jobs to the controller via the HTTP API, which then launches and maintains a cluster of cloud VMs, and maintains the job queue and metadata in a local database.

Our service integrates, and interfaces with two primary services. First, it uses the Google cloud API [?] for launching, terminating, and monitoring VMs. Once a cluster is launched, it then configures a cluster manager such as Slurm [?] or Torque [?], to which it submits jobs. Our service uses the Slurm cluster manager, with each VM acting as a Slurm "cloud" node, which allows Slurm to gracefully handle VM preemptions. The Slurm master node runs on a small, 2 CPU non-preemptible VM, which is shared by all applications and users. We monitor job completions and failures (due to VM preemptions) through the use of Slurm call-backs, which issue HTTP requests back to the central service controller.

Policy Implementation: Our service creates and manages clusters of transient cloud servers, manages all aspects of the VM lifecycle and costs, and implements the model-based policies. It manages a cluster of VMs, and parametrizes the bathtub model based on the VM type, region, time-of-day, and day-of-week. When a new batch job is to be launched, it finds a "free" VM in the cluster that is idle, and uses the job scheduling policy to determine if the VM is suitable or a new VM must be launched. Due to the bathtub nature of the failure rate, VMs that have survived the initial failures are "stable" and have a very low rate of failure, and thus are "valuable". We keep these stable VMs as "hot spares" instead of terminating them, for a period of one hour. For the checkpointing policy, our dynamic programming algorithm has a time complexity of $O(T^3)$, for a job of length T. To minimize this overhead, we precompute

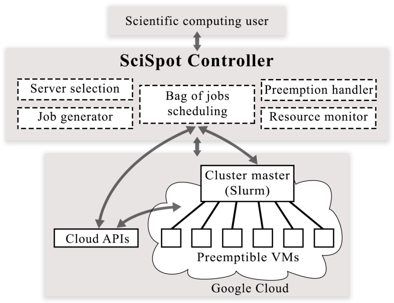


Figure 4: Architecture and system components of our batch computing service.

the checkpointing schedule of jobs of different lengths, and don't need to compute the checkpoint schedule for every new job.

Bag of Jobs Abstraction For Scientific Simulations: While our service is intended for general batch jobs, we incorporate a special optimization for scientific simulation workloads that improves the ease-of-use of our service, and also helps in our policy implementation. Our insight is that most scientific simulations involve launching a series of jobs that explore a large parameter space that results from different combinations of physical and computational parameters.

These workloads therefore can be abstracted as a "bag of jobs", with each job running the same application with slightly different parameters. A bag of jobs is characterized by the job and all the different parameters with which it must be executed. Within a bag, jobs show little varition in their running time and execution characteristics.

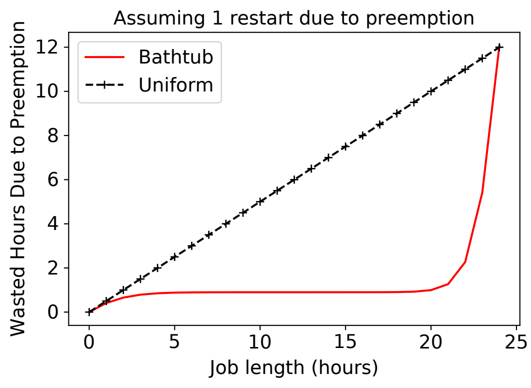
We allow users to submit entire bags of jobs, which permits us to determine the running time of the jobs, which we use in our scheduling and checkpointing policy. Having a large sequence of jobs is also particularly useful with bathtub preemptions, since we can re-use "stable" VMs with low preemption probability for running new jobs from a bag. If jobs were submitted one at a time, a batch computing service may have to terminate the VM after job completion, which would increase the job failure probability resulting from running on new VMs that have a high initial failure rate.

6 MODEL AND POLICY EVALUATION

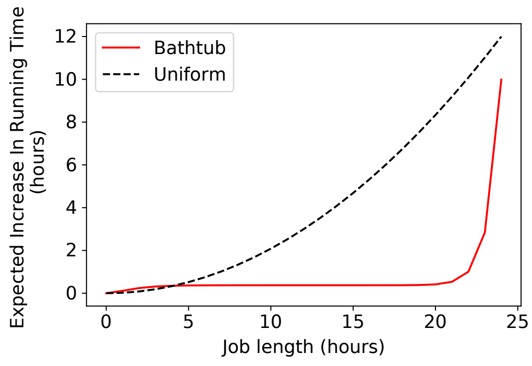
In this section, we present analytical and empirical evaluation of constrained preemptions. We have already presented the statistical analysis of our model in Section **??**, and we now focus on answering the following questions:

(1) How do constrained preemptions impact the total running time of applications?
(2) How What is the effect of our model-based policies when compared to existing transient computing approaches?
(3) What is the cost and performance of our batch computing service for real-world workloads?

Environment and Workloads: All our empirical evaluation is conducted on the Google Public cloud using our batch computing service described in the previous section. We use three scientific computing workloads that are representative of typical applications in the broad domains of physics, material sciences, and chemical engineering:



(a) Computation wasted due to one preemption.



(b) Expected increase in running time.

Figure 5: Wasted computation and expected increase in running time for uniform vs. baththub failures. For jobs $>$ 5 hours, bathtub distribution results in significantly lower wasted computation.

Nanoconfinement. The nanoconfinement application launches molecular dynamics (MD) simulations of ions in nanoscale confinement created by material surfaces [? ?].
Shapes. The Shapes application runs an MD-based optimization dynamics to predict the optimal shape of deformable, charged nanoparticles [? ?].
LULESH. Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) is a popular benchmark for hydrodynamics simulations of continuum material models [? ?].

All applications are run with default parameters unless otherwise stated.

6.1 Impact of Constrained Preemptions on Job Running Times

We begin by examining how constrained preemptions impacts the total job running times. When a preemption occurs during the job's execution, it results in wasted work, assuming there is no checkpointing. This increases the job's total expected running time, since it must restart after a preemption.

The expected wasted work depends on two factors:

(1) The probability of the job being preempted during its execution.
(2) *When* the preemption occurs during the execution.

For a job of length J, the wasted work, assuming that the job faces a *single* preemption, is $E[W_1(J)]$, and is given by Equation **??**. We first analyze this wasted work for jobs of different lengths in Figure 5a We analyze two failure probability distributions for constrained preemptions: a uniform distribution such that $F(t) = 24-t$, and the bathtub shaped distribution with parameters corresponding to the `n1-highcpu-16` VM type shown in Figure **??**.

For the uniform distribution, the wasted work is linear in the job length, and is given by $J/2$. For the bathtub distribution, the wasted work is given by Equation **??**, and is *significantly* lower, especially for longer jobs (longer than 5 hours). With the bathtub distribution, jobs see a high rate of failure initially, but that also reduces the wasted work. Once jobs survive the initial high failure rate, the rate of failure is low, and thus the wasted work is more or less constant for all but the shortest and longest jobs.

We now examine the expected increase in running time, that also accounts for the probability of failure, and is given by $P(\text{failure}) * E[W_1]$. Figure 5b shows this expected increase in running times for jobs of different lengths. We see that for uniformly distributed preemptions, the increase in running time is quadratic in the job length (and is given by $J^2/48$). Interestingly, the high rate of early failures for the bathtub distribution results in a slightly worse (i.e., higher) running time for short jobs. However for jobs longer than 5 hours, a cross-over point is reached, and the bathtub distribution provides a significantly lower overhead of preemptions. For instance, for a 10 hour job, the increase in running time is about 30 minutes, or 5%. In contrast, if failures were uniformly distributed, the increase would be 2 hours.

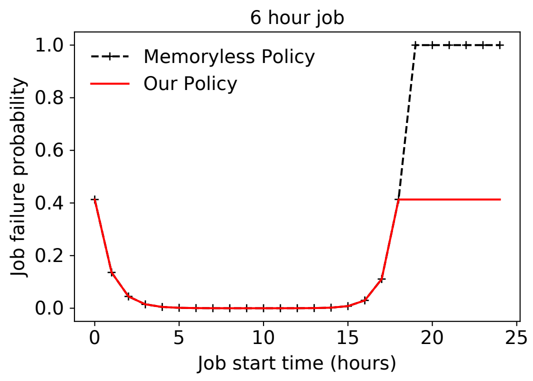
Thus, the bathtub preemptions are beneficial for applications and users, as the low failure rate during the middle periods results in significantly lower wasted work, compared to the uniformly distributed failures. Since the failure rate distribution is ultimately controlled by the cloud provider, our analysis can be used to determine the appropriate distribution based on the job length distributions. For instance, if short jobs are very common, then uniformly distributed preemptions are preferable, otherwise, bathtub distributions can offer significant benefits.
Result: *For constrained preemptions, bathtub distributions significantly reduce the expected increase in running times for medium to long running jobs, but are slightly inferior for short jobs.*

6.2 Model-based Policies

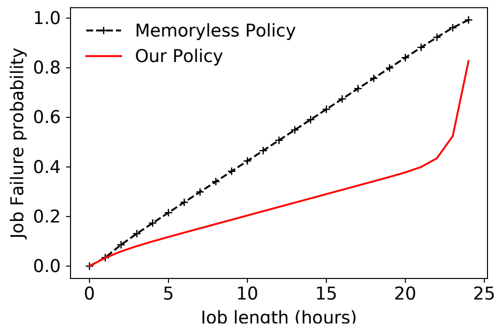
We now evaluate the effectiveness of model-driven policies that we proposed earlier in Section **??**. Specifically, we seek to compare the effectiveness of our job scheduling and checkpointing policies with existing transient computing approaches.

6.2.1 Job Scheduling. In the previous subsection, we have quantified the increase in running time due to preemptions, but we had assumed that jobs start on a newly launched server. In many scenarios however, a server may be used for running a long-running sequence of jobs, such as in a batch-computing service.

Our job scheduling policy is model-driven and decides whether to request a new VM for a job or run it on an existing VM. A new



(a) Effect of job start time on the failure probability.



(b) Job failure probability for jobs of different lengths.

Figure 6: Job failure probability is lower with our deadline aware policy across all job sizes.

VM may be preferable if the job starts running near the VM's 24 hour preemption deadline.

Figure 6a shows the effect of our job scheduling policy for an 8 hour job, for different job starting times (relative to the VM's starting time). We compare against a baseline of memoryless job scheduling that is not informed by constrained preemption dynamics. Such memoryless policies are the default in existing transient computing systems such as SpotOn [?] and others []. In the absence of insights about bathtub preemptions, the memoryless policy continues to run jobs on the existing VM. As the figure shows, the empirical job failure probability is bathtub shaped. However since the job is 8 hours long, with the memoryless policy, it will always fail when launched after $24 - 8 = 16$ hours. In contrast, our model-based policy determines that after 16 hours, we will be better off running the job on a newer VM, and results in a much lower job failure probability (=0.4). Thus, our model-based job scheduling policy can reduce job failure probability by taking into account the time-varying failure rates of VMs, which is not considered by existing systems that use memoryless scheduling policies.

The job failure probability is determined by the job length and the job starting time. We examine the failure probability for jobs of different lengths in Figure 6b, in which we average the failure probability across different start times. We again see that our policy results in significantly lower failure probability compared to memoryless scheduling. For all but the shortest and longest jobs, the failure probability with our policy is *half* of that of existing memoryless policies. This difference is primarily due to the differences in how the two policies perform for jobs launched near the end of the VM preemption deadline, which we examined previously in Figure 6a.

Result: *Our model-based job scheduling policy can decrease job failure probability by* 2×.

6.2.2 Checkpointing. We now evaluate our model-based checkpointing policy, that uses a dynamic programming approach. With our policy, the checkpointing rate is determined by the VM's current failure rate. In contrast, all prior work in transient computing and most prior work in fault-tolerance assumes that failures are exponentially distributed (i.e., memoryless), and use the Young-Daly checkpointing interval. In the Young-Daly approach, checkpoints are taken after a constant period given by $\tau \propto \sqrt{MTTF}$. However in the case of constrained preemptions with bathtub distributions, the failure rate is time-dependent and not memoryless.

The expected increase in running time for a 4 hour job is shown in Figure 7a, in which we account for both the increase due to the checkpointing overhead, as well as the expected recomputation due to preemptions. Throughout, we assume that each checkpoint takes 1 minute. The increase in running time depends on the failure rate and thus the job's starting time. With our model-based checkpointing policy, the increase in running time is bathtub shaped and is below 5%, and around 1% when the job is launched when the VM is between 5 and 15 hours old.

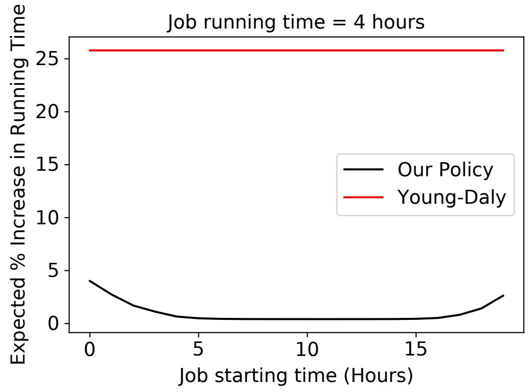
We also compare with the Young-Daly periodic checkpointing policy, and use the initial failure rate of the VM to set the MTTF, which corresponds to an MTTF of 1 hour. This results in a high, constant rate of checkpointing, and thus increases the running time of the job by more than 25%. The increase in running time is primarily due to the overhead of checkpointing. Note that checkpointing with a lower frequency decreases the checkpointing overhead, but increases the recomputation required.

Next, we examine the expected running time of jobs of different length, when all jobs start at time=0, i.e, are launched on a freshly launched VM. Figure 7b shows the expected increase in the running time of the jobs with our model-based checkpointing policy and the Young-Daly policy with MTTF=1 hour. With our policy, the running times increase by 10% for short jobs less than 2 hours long, and increase by less than 5% for longer jobs. In contrast, the Young-Daly policy yields a constant increase in running times of 25%. Thus, our model-based policy is able to reduce the checkpointing overhead and thus reduce the performance overhead of running on preemptible VMs to below 5%.

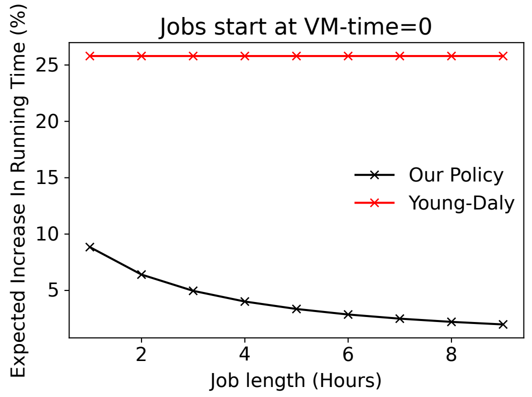
Result: *Our checkpointing policy can reduce the performance overhead of preemptions to under 5%, which is* 5× *better than conventional Young-Daly policies.*

6.3 Effectiveness on Scientific Computing Workloads

We now show the effectiveness of our batch computing service on Google Preemptible VMs. We run scientific simulation workloads described earlier in this section, and are interested in understanding



(a) Checkpointing overhead for jobs of length 4 at different starting times



(b) Increase in running time with Checkpointing when jobs start at time=0.

Figure 7: Checkpointing effectiveness.

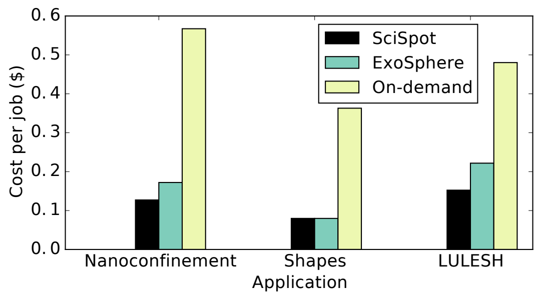


Figure 8: SciSpot's use of preemptible VMs can reduce costs by up to 5× compared to conventional cloud deployments, and 20% compared to the state of the art EC2 spot instance selection (ExoSphere [?]).

the real-world effectiveness of our model-based service. We use our model-driven job scheduling policy, but do not use checkpointing, since it requires additional

Cost: The primary motivation for using preemptible VMs is their significantly lower cost compared to conventional "on-demand" cloud VMs that are non-preemptible. To evaluate the cost of using our batch computing service, we run a bag of 32 jobs, all running on a cluster of 32 VMs of type n1-highcpu-32. Within a bag, different jobs are exploring different physical parameters, and job running times show little variance. Figure 8 shows the cost of using Preemptible VMs compared to conventional on-demand VMs. We see that for all the three applications, using our service can reduce costs by 5×.

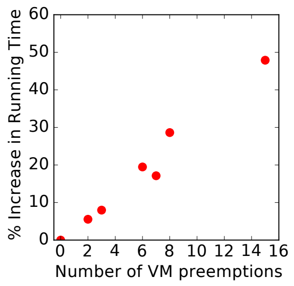


Figure 9: The increase in running time due to preemptions is under 50%, even at high preemption rates.

We note that for this experiment, our service was using model-driven job scheduling, but was not using checkpointing, since the applications lacked checkpointing mechanisms. However, incorporating checkpointing would reduce the costs even further, since it would reduce the increase in running time (and server costs) due to recomputation.

Preemptions: Finally, we examine the effect of preemptions on the increase in running time under real-world settings. We ran a cluster of 32 n1-highcpu-32 VMs running the Nanoconfinement application, and repeated the experiment multiple times to observe the effect of preemptions. Figure 9 shows the increase in running time of the entire bag of jobs, when different number of VM preemptions are observed during the entire course of execution. We see that the net impact of preemptions results in a roughly linear increase in running time. Each preemption results in a roughly 3% increase in running time, which validates our analytical evaluation shown earlier in Figure 5b.

Result: *Our batch computing service can reduce costs by up to 5× compared to conventional on-demand cloud deployments. In practice, the performance impact of preemptions is as low as 3%.*

7 RELATED WORK

7.1 Transient Cloud Computing

The challenges posed by Amazon EC2 spot instances, the first transient cloud servers, have received significant attention from both academia and industry [?]. The significantly lower cost of spot instances makes them attractive for running preemption and delay tolerant batch jobs [? ? ? ? ? ? ?]. The distinguishing characteristic of EC2 spot instances is their dynamic auction-based pricing, and choosing the "right" bid price to minimize cost and performance degradation is the focus of much of the past work on transient computing [? ? ? ? ? ? ? ? ? ? ?]. However, as explained in Section ??, it remains to be seen how Amazon's recent change [?] in the preemption model of spot instances affects prior work.

On the other hand, the effective use of transient resources provided by other cloud providers such as Google, Microsoft, Packet, and Alibaba largely remains unexplored. Ours is the first work that studies the preemption characteristics and addresses the challenges involved in running large-scale applications on the Google Preemptible VMs, and provides insights on the unique preemption dynamics.

7.1.1 Preemption Mitigation. Effective use of transient servers usually entails the use of fault-tolerance techniques such as checkpointing [**?**], migration [**?**], and replication. In the context of HPC workloads, [**? ? ?**] develop checkpointing and bidding strategies for MPI applications running on EC2 spot instances, based on older spot pricing models.

Periodic checkpointing [**?**] is not optimal in our case because preemptions are not memoryless. By treating bags of jobs as an execution unit, allowing some jobs to fail, and using insights from preemption models, we show that it is possible to reduce the recomputation times to acceptable levels even without the use of periodic checkpointing that imposes additional deployment and performance overheads. Our preemption model for Google preemptible VMs developed in Section 3 provides a novel characterization of bathtub shaped failure rates not captured by the classic Weibull distribution, and is distinct from prior efforts [**? ?**].

7.1.2 Fault Tolerance.

8 CONCLUSION

Given the rise of transient cloud computing and its use in web services and distributed data processing, it is not the question of if, but when, transient cloud computing becomes a credible and powerful alternative to HPC clusters for scientific computing applications. In this paper, we developed principled approaches for deploying and orchestrating scientific computing applications on the cloud, and presented Our service, a framework for low-cost scientific computing on transient cloud servers. Our service develops the first empirical and analytical preemption model of Google Preemptible VMs, and uses the model for mitigating preemptions for “bags of jobs”. Our service’s cost-minimizing server selection and job scheduling policies can reduce costs by up to 5× compared to conventional cloud deployments. When compared to HPC clusters, Our service can reduce the total job turnaround times by up to 10×.